LA-UR-25-23479

Approved for public release; distribution is unlimited.

Title: Driving Continuous Integration and Developer Workflows with Spack

Author(s): Berger, Richard Felix

Intended for: Spack User Meeting, 2025-05-07/2025-05-08 (Chicago, Illinois, UNITED STATES)

Issued: 2025-04-10









Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by Triad National Security, LLC for the National Nuclear Security Administration of U.S. Department of Energy under contract 89233218CNA000001. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness. UNCLASSIFIED



Driving Continuous Integration and Developer Workflows with Spack

Richard Berger May 8, 2025



UNCLASSIFIED



Outline

Motivation

Driving CMake with Spack

Spack Deployments



Outline

Motivation

Driving CMake with Spack

Spack Deployments



What are we building and testing?

- C/C++/Fortran HPC codes
- Research and Production codes
- Run on Tri-Lab (LANL, LLNL, Sandia) systems
 - LANL: Crossroads / Rocinante / Chicoma / Venado
 - LLNL: El Capitan / Tuolumne / Sierra
- CPU-only and GPU-accelerated systems







What are we building and testing?

- Builds: Release, Debug
- Code Variants / Library Configurations
- Compilers: GCC, Clang, Intel, NVIDIA, AMD
- Accelerations: OpenMP, CUDA, ROCm
- Communication:
 - MPI (MPICH, OpenMPI, CrayMPICH)
 - GASNET
- Hardware
 - GPUs: NVIDIA Volta/Ampere, AMD MI250/MI300A
 - Interconnects: HPE/Cray Slingshot 11, Infiniband



prechecke

format

Continuous Integration (CI) Pipelines

- GitLab for Source Code Management
- Pipelines for CI
 - main branch
 - scheduled (nightly, weekly)
 - merge requests
- Jacamar CI Runners for TriLab systems
 - jobs can run via batch system



rzvernal_mpi_rocm_cravmpich_qfx9...





prechecks

format

CI Failures: What now?

\bigotimes	LOS Alamos
--------------	------------





rzvernal_mpi_rocm_cravmpich_qfx9...

5/8/2025 | 7

CI Failures: What now?

Developers "How do I recreate that CI job?"







rzvernal_mpi_rocm_craympich_qfx9...

darwin legion cuda mnich amnere darwin legion cuda mpich ampere darwin legion cuda mnich volta de ... A darwin lagion cuda mnich volta ral darwin legion openmp mpich debug darwin legion openmp mpich nomi... darwin legion openmo moich nomi... darwin legion openmin mnich release darwin legion room mpich afy90a darwin legion rocm mpich gfx90a darwin legion rocm mpich gfx90a ... darwin legion rocm mpich gfx90a rocipante legion openmo cravmoic rocinante_legion_openmp_cravmpic... a recipante legion openmo craympic rocinante legion openmo craympic... rocinante legion serial cravmpich i... rocinante legion serial craympich i rządams legion rocm craymnich of ... a rzadams legion rocm craympich of rzvernal legion rocm cravmpich gf... rzvernal_legion_rocm_cravmpich_gf...

UNCLASSIFIED

Our answer

ssh rzadams
cd /your/code/checkout
flux alloc -N=1 --time-limit=2h
source .gitlab/build_and_test.sh rzadams craympich-rocm-gfx942-gcc



Outline

Motivation

Driving CMake with Spack

Spack Deployments



















Requirement: Spack package for our code

```
class MyPackage(CMakePackage):
```

git = "https://github.com/namespace/myproject.git"

```
version("main", branch="main")
```

```
variant("boost", default=False)
variant("kokkos", default=False)
```

```
depends_on("cmake@3.20:", type="build")
depends_on("boost@1.80:", when="+boost")
depends_on("kokkos@4:", when="+kokkos")
depends_on("googletest@1.12:", type="test")
```

- How to get the source code
- Which versions exist
- Which configuration options exist
- What are dependencies, given these configuration options and other factors



. . .

Requirement: Spack package for our code

```
Given a build configuration
                                                         and an environment, what
class MyPackage(CMakePackage):
                                                         are the options for our build
  . . .
                                                         system?
 def cmake_args(self):
    args = [
     self.define_from_variant("ENABLE_BOOST", "boost"),
     self.define_from_variant("ENABLE_KOKKOS", "kokkos"),
      self.define("ENABLE_CUDA", self.spec.satisfies("^kokkos+cuda")),
     self.define("BUILD_TESTING", self.run_tests),
    ٦
```

return args



Spack Environments = Environment + Build Configuration

```
# minimal/spack.yaml
spack:
   spec:
    - mypackage
   packages:
      mypackage:
      require:
      require:
```

- "+boost"

- # ampere/spack.yaml
 spack:
 spec:
 mypackage
 - packages:
 - mypackage:
 - require:
 - "+boost +kokkos"
 - kokkos:
 - require:
 - "+cuda cuda_arch=80"



Installing our code with Spack

activate environment defined in myenu/spack.yaml
spack env activate -d myenv

install our software
spack install

A Note: This is **not** what we want for CI and Development!



spack env activate -d myenv

```
spack develop -p ${SOURCE_DIR} -b ${BUILD_DIR} --no-clone myproject@main
spack concretize -f
spack install --include-build-deps --only dependencies
```

spack install --test root --until cmake myproject

Spack Environment



```
spack env activate -d myenv
```

```
spack develop -p ${SOURCE_DIR} -b ${BUILD_DIR} --no-clone myproject@main
spack concretize -f
spack install --include-build-deps --only dependencies
```

spack install --test root --until cmake myproject





```
spack env activate -d myenv
```

```
spack develop -p ${SOURCE_DIR} -b ${BUILD_DIR} --no-clone myproject@main
spack concretize -f
spack install --include-build-deps --only dependencies
```

spack install --test root --until cmake myproject





```
spack env activate -d myenv
```

```
spack develop -p ${SOURCE_DIR} -b ${BUILD_DIR} --no-clone myproject@main
spack concretize -f
spack install --include-build-deps --only dependencies
```

spack install --test root --until cmake myproject





Getting the Spack Build Environment

interactive sub-shell
spack build-env myproject bash

sourceable shell script

spack build-env --dump \${BUILD_DIR}/build_env.sh myproject





Using the Spack Build Environment

activate build environment
source \${BUILD_DIR}/build_env.sh

undo Spack CMake default of verbose Makefiles
cmake -DCMAKE_VERBOSE_MAKEFILE=off \${BUILD_DIR}

build code
cmake --build \${BUILD_DIR} --parallel



CI script helpers

```
# build code
cmake --build ${BUILD_DIR} --parallel
```



CI script helpers

```
# simplified short versions of helpers in build_and_test.sh
prepare_env() {
 spack activate -d ${SPACK ENV NAME}
 spack develop -p $SOURCE_DIR -b ${BUILD_DIR} --no-clone myproject@main
 spack concretize -f
 spack install --include-build-deps --only dependencies -j $(nproc) myproject
spack_cmake_configure() {
 spack install --test root -u cmake myproject
 spack build-env --dump ${BUILD_DIR}/build_env.sh myproject
```



CI Job Steps



- · each step is a bash shell function and can be invoked manually
- CI just goes through all of them in a fixed order
- developers's can stop at any point, make modifications, and continue via the shell functions.



CI Job Steps



- each step is a bash shell function and can be invoked manually
- · CI just goes through all of them in a fixed order
- developers's can stop at any point, make modifications, and continue via the shell functions.



Outline

Motivation

Driving CMake with Spack

Spack Deployments



Spack Deployments

Deployment

Spack Checkout	
Configuration	
Environments (no view)	
Installed Packages	

network filesystem





spack:

include:

- \$spack/../systems/common/packages.yaml
- \$spack/../systems/common/repos.yaml
- \$spack/../systems/ATS4/common/compilers.yam1
- \$spack/../systems/ATS4/common/packages.yaml
 specs:

specs:

- myproject

view: true

concretizer:

unify: true

packages:

kokkos:

require:

- "+rocm amdgpu_target=gfx942"



spack:

include:

project-specific configuration

- \$spack/../systems/common/packages.yaml
- \$spack/../systems/common/repos.yam1
- \$spack/../systems/ATS4/common/compilers.yaml
- \$spack/../systems/ATS4/common/packages.yam1

specs:

- myproject
- view: true
- concretizer:
 - unify: true
- packages:
 - kokkos:

require:

- "+rocm amdgpu_target=gfx942"



spack:

include:

- \$spack/../systems/common/packages.yaml
- \$spack/../systems/common/repos.yaml
- \$spack/../systems/ATS4/common/compilers.yam1
- \$spack/../systems/ATS4/common/packages.yaml

specs:

- myproject system-specific configuration

view: true concretizer: unify: true packages: kokkos: require: - "+rocm amdgpu_target=gfx942"



spack:

include:

- \$spack/../systems/common/packages.yaml
- \$spack/../systems/common/repos.yaml
- \$spack/../systems/ATS4/common/compilers.yam1
- \$spack/../systems/ATS4/common/packages.yaml

specs:

- myproject
- view: true

concretizer:

unify: true

packages: kokkos:

environment-specific configuration

require:

- "+rocm amdgpu_target=gfx942"

Spack Deployments

Deployment

Spack Checkout
Configuration
Environments (no view)
Installed Packages

network filesystem



Spack Deployments



network filesystem

compute node (tmpfs)



Bootstrap & Spack Mirrors



spack:

include:

- \$spack/../systems/common/packages.yaml
- \$spack/../systems/common/repos.yaml
- \$spack/../systems/ATS4/common/bootstrap.yaml # generated, pointing to original
- \$spack/../systems/ATS4/common/compilers.yaml
- \$spack/../systems/ATS4/common/packages.yaml
- \$spack/../systems/ATS4/common/upstreams.yaml # generated, pointing to original
- \$spack/../systems/ATS4/common/mirrors.yaml # generated, pointing to original
 specs:
 - myproject

view: true

concretizer:

unify: true

packages:



Optimizing Replication



Deploying on air-gapped systems



What Developers and the CI see





Summary

Deployment Pipeline

- places read-only shared Spack instance at known location at each target system
- create bootstrap and package source mirrors for all environments
- prebuilds software dependencies of all environments on all target systems

Continuous Integration Pipelines / Developer Workflows

- utility script drives entire build and test process for both CI and Development
- script creates a (temporary) Spack instance that uses a deployment as upstream
- Spack environments are used to encode and create the build environment and build configuration



UNCLASSIFIED

Q Questions

